Kingswap NFT Smart Contract AUDIT REPORT

Prepared By :

Celticlab Private Limited

Corporate Office:

H. No : 45

Awas Vikas Colony

Basti

U.P

Pin : 272001

India

CIN – U72900UP2017PTC097710

Email ID - himanshu@celticlab.com

Introduction

This is a technical audit for KingToken_NFT smart contracts. This document outlines our methodology, limitations and results for our security audit. The solidity files covered are :

1. KingTokenNFTs_King.sol
2. KingTokenNFTs_Knight.sol
3. KingTokenNFTs_Queen.sol

All activities were conducted in a way which aimed to simulate the mindset of a malicious actor engaging in a targeted attack towards the above mentioned smart contracts with the expected goals of:

• Identifying ways to manipulate the state modifying logic of the smart contract.
• Shedding light on bad coding practices.
• Determining the impact of external malicious behaviour.

Due to the nature of smart contracts, the importance of security cannot be overstated. Once a smart contract goes live, it will be very challenging to correct any mistakes. This furthermore enhances the importance of auditing code, skipping this step, raises a great risk of potential value loss.

Synopsis

In regards to modularity and simplicity, it is a good practice to keep contracts and functions small, and in the concerned smart contracts the such practices are followed. It is recommended to prefer clarity over performance whenever possible and use existing code from contracts such as the ones provided by OpenZeppelin which is thoroughly audited and tested, anywhere possible. OpenZeppelin contracts are used in the implementation. Overall, the code demonstrates high code quality standards adopted and effective use of concept and modularity. The contract development team demonstrated high technical capabilities, both in the design of the architecture and in the implementation.

Code Analysis

Besides, the results of the automated analysis, manual verification was also taken into account. The complete contract was manually analyzed, every logic was checked and compared with the comments made in the contract. The manual analysis of code confirms that the Contract does not contain any serious susceptibility. No divergence was found between the logic in Smart Contract and the informative smart contract comments.

Scope

This audit is into the technical and security aspects of the KingToken_NFT smart contracts. The key aim of this audit is to ensure that transactions taking place in these contracts are not by far attacked or misused by any third party and they occur in the right intention of the transaction initiator. The next aim of this audit is to ensure the coded algorithms work as expected. The audit of Smart Contract also checks the implementation of

token mechanism i.e. The KingToken_NFT contract must follow the ERC721 Standard.
This audit is purely technical and is not investment advice. The scope of the audit is limited to the below source codes:
1. KingTokenNFTs_King.sol

```
pragma solidity ^0.6.0;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/v3.1.0/contracts/token/ERC721/ERC721.sol";
import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/v3.1.0/contracts/utils/Counters.sol";

contract KingTokenNFTKing is ERC721 {

    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;

    //Owner address
    address private owner;

    //List of user address;
    mapping(address => uint) private wallets;

     modifier _onlyOwner(){
        require(msg.sender == owner);
        _;
    }


    constructor(string memory myName, string memory mySymbol,address _owner) ERC721(myName, mySymbol)
public {
        owner = _owner;
    }

    //for owner to see NFTs issued to who
    function checkWalletAddressExist(address player) _onlyOwner() public view returns (uint){
        return wallets[player];
    }

    function mint(address player, uint totalSupply) _onlyOwner() public returns (bool) {

        for(uint256 i = 1; i<= totalSupply; i++){
            _tokenIds.increment();
            uint256 newItemId = _tokenIds.current();
            _mint(player, newItemId);
            _setTokenURI(newItemId,"https://api.jsonbin.io/b/5f7367137243cd7e82466615/31");
        }
        //check if address exist in wallet
        if(wallets[player] == 0){
             wallets[player] = totalSupply;
        }else{
                uint totalsum = wallets[player];
                totalsum = totalsum + totalSupply;
                wallets[player] = totalsum;
        }
    }

}
```

2. KingTokenNFTs_Knight.sol

```
pragma solidity ^0.6.0;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/v3.1.0/contracts/token/ERC721/ERC721.sol";
import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/v3.1.0/contracts/utils/Counters.sol";

contract KingTokenNFTKnight is ERC721 {

    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;

    //Owner address
    address private owner;

    //List of user address;
    mapping(address => uint) private wallets;

     modifier _onlyOwner(){
        require(msg.sender == owner);
        _;
    }


    constructor(string memory myName, string memory mySymbol,address _owner) ERC721(myName, mySymbol)
public {
        owner = _owner;
    }

    //for owner to see NFTs issued to who
    function checkWalletAddressExist(address player) _onlyOwner() public view returns (uint){
        return wallets[player];
    }

    function mint(address player, uint totalSupply) _onlyOwner() public returns (bool) {

        for(uint256 i = 1; i<= totalSupply; i++){
            _tokenIds.increment();
            uint256 newItemId = _tokenIds.current();
            _mint(player, newItemId);
            _setTokenURI(newItemId,"https://api.jsonbin.io/b/5f7367137243cd7e82466615/30");
        }
        //check if address exist in wallet
        if(wallets[player] == 0){
              wallets[player] = totalSupply;
        }else{
                uint totalsum = wallets[player];
                totalsum = totalsum + totalSupply;
                wallets[player] = totalsum;
        }
    }

}
```

3. KingTokenNFTs_Queen.sol

```
pragma solidity ^0.6.0;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/v3.1.0/contracts/token/ERC721/ERC721.sol";
import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/v3.1.0/contracts/utils/Counters.sol";

contract KingTokenNFTQueen is ERC721 {

    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;

    //Owner address
    address private owner;

    //List of user address;
    mapping(address => uint) private wallets;

     modifier _onlyOwner(){
        require(msg.sender == owner);
        _;
    }


    constructor(string memory myName, string memory mySymbol,address _owner) ERC721(myName, mySymbol)
public {
        owner = _owner;
    }

    //for owner to see NFTs issued to who
    function checkWalletAddressExist(address player) _onlyOwner() public view returns (uint){
        return wallets[player];
    }

    function mint(address player, uint totalSupply) _onlyOwner() public returns (bool) {

        for(uint256 i = 1; i<= totalSupply; i++){
            _tokenIds.increment();
            uint256 newItemId = _tokenIds.current();
            _mint(player, newItemId);
            _setTokenURI(newItemId,"https://api.jsonbin.io/b/5f7367137243cd7e82466615/29");
        }
        //check if address exist in wallet
        if(wallets[player] == 0){
              wallets[player] = totalSupply;
        }else{
                uint totalsum = wallets[player];
                totalsum = totalsum + totalSupply;
                wallets[player] = totalsum;
        }
    }

}
```

Testing

Primary checks followed during testing of Smart Contract is to see that if code :

- We check the Smart Contracts Logic and compare it with the standards, check the gas usage optimization and the implemented business logic.
- The contract code should follow the Conditions and logic as per user request.
- We deploy the Contract and run the Tests.
- We make sure that the Contract does not lose any money/Ether.

Known Vulnerabilities Check

Smart Contract was scanned for commonly known and more specific vulnerabilities.

Following are the list of commonly known vulnerabilities that was considered during the (manual and automated) audit of the smart contract and their comments:

1. TimeStamp Dependence : The timestamp of the block can be manipulated by the miner, and so should not be used for critical components of the contract. If required, *Block numbers* and *average block time* can be used to estimate time.

Comments : KingToken_NFT smart contracts do not have any timestamp dependence in their code.

2. Gas Limit and Loops : Sometimes the number of iterations in a loop can cause the transaction gas limit to grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Hence, the loops must be used in a very calculative fashion.

Comments : KingToken_NFT smart contracts are free from the gas limit check as the contracts do not contain any loop in their code.

3. Compiler Version : Contracts should be deployed with the same compiler version and flags that they have been tested the most with. Locking the pragma helps ensure that contracts do not accidentally get deployed to an upgraded compiler version, as future compiler versions may handle certain language constructions in a way the developer did not foresee.

Comments : Vulnerability Level: Low

In KingToken_NFT smart contracts the compiler version is not fixed.

4. ERC721 Standards : KingToken_NFT smart contract follows all the universal ERC721 coding standards and implements all its functions and events in the contract code.

5. Redundant fallback function : The standard execution cost of a fallback function should be less than 2300 gas.

Comments : KingToken_NFT smart contracts do not have any fallback function, hence they are free from this vulnerability.

6. Unchecked math : Need to guard uint overflow or security flaws by implementing the proper maths logic checks.

Comments: Vulnerability level : Low

The KingToken_NFT smart contracts do not use the popular SafeMath library for critical operations to avoid arithmetic over or underflow and safeguard against unwanted behaviour. It is recommended to use the SafeMath library.

7. Exception disorder : When an exception is thrown, it cannot be caught: the execution stops, the fee is lost. The irregularity in how exceptions are handled may affect the security of contracts.

Comments : In manual testing of KingToken_NFT smart contracts, there are no exception disorders found due to code or syntax issues.

8. Reentrancy : The reentrancy attack consists of the recursively calling a method to extract ether from a contract if the user is not updating the balance of the sender before sending the ether.

Comments : There is no risk of reentrancy attack in KingToken_NFT smart contracts as there are no calls to external functions for sending ether.

9. DoS with (Unexpected) Throw : The Contract code can be vulnerable to the Call Depth Attack! So instead, code should have a pull payment system instead of push.

Comments : The KingToken smart contracts do not implement any payment related scenario thus not vulnerable to this attack.

10. DoS with Block Gas Limit : In a contract by paying out to everyone at once, contract risk running into the block gas limit. Each ethereum block can process a certain maximum amount of computation. If one tries to go over that, the transaction will fail. Therefore again push over pull payment is suggested to remove the above issue.

Comments : The KingToken_NFT smart contracts do not implement any payment related scenario thus not vulnerable to this attack.

11. Explicit Visibility in functions and state variables : Explicit visibility in the function and state variables are provided. Visibility like external, internal, private and public is used and defined properly.

Comments : There are variables in the KingToken_NFT smart contracts which are declared with "private" modifiers. We would like to point out a popular misconception among developers that "private" modifiers make a variable invisible to the outside world (of smart contract). However, the miners or investors have view access to all of the contract's code, data or state changes.

Automated test results

We have used the truffle framework and smart check tool for writing and generating automated test results.

Below are the ERC721 test cases checked :

- Should have correct "name" definition - passed

- Should have correct "totalSupply" definition - passed - Should have correct "symbol" definition - passed

- Should have correct "balanceOf" definition - passed

- Should have correct "approve" definition - passed

- Should have correct "Transfer" definition - passed

- The deployer of the contract is the owner of the contract - passed

- Should generate an ERC721 token with proper configuration - passed

- Two randomly generated ethereum addresses are able to receive tokens and transfer - passed

- Should throw error on trying to transfer wrong token id - passed

- An address cannot transfer tokens that it doesn’t have - passed

- Should allow an address to approve another address for transferring tokens - passed

- Should allow an address to zero out the previously allowed amount - passed

- Should not allow any address to send tokens to 0x0 address - passed

- Should not allow spender address to send more tokens than it has been approved to send - passed

- Event “Transfer” should log as expected - passed

- Event “Approval” should log as expected - passed

AUTOMATED TEST RESULTS -

Passed - 16

Failed - 0

Risk

NO CRITICAL RISKS FOUND

The KingToken_NFT Smart Contracts have no risk of losing any amounts of ether/tokens in case of external attack or a bug, as the contracts do not take any kind of funds from the user as investment. If anyone tries to send any amount of ether to the contract address, the transaction will cancel itself and no ether comes to the contracts.

Conclusion and Results

In this report, we have concluded about the security standards of KingToken_NFT Smart Contracts. The smart contract has been analysed under different facets. Code quality is very good, and well modularised. We found that KingToken_NFT smart contract adapts a very good coding practice and has clean, documented code. No severe discrepancies were found.